

Other GPU programming techniques



Fernanda Foertter

foertterfs@ornl.gov



U.S. DEPARTMENT OF
ENERGY



OAK RIDGE NATIONAL LABORATORY

MANAGED BY UT-BATTELLE FOR THE DEPARTMENT OF ENERGY

Outline

- Accelerators, now what?
- Programming models
- Using GPU libraries
- Accelerator frameworks
- Coding for the hardware
- Conclusions

Accelerators

- Accelerators, now what?



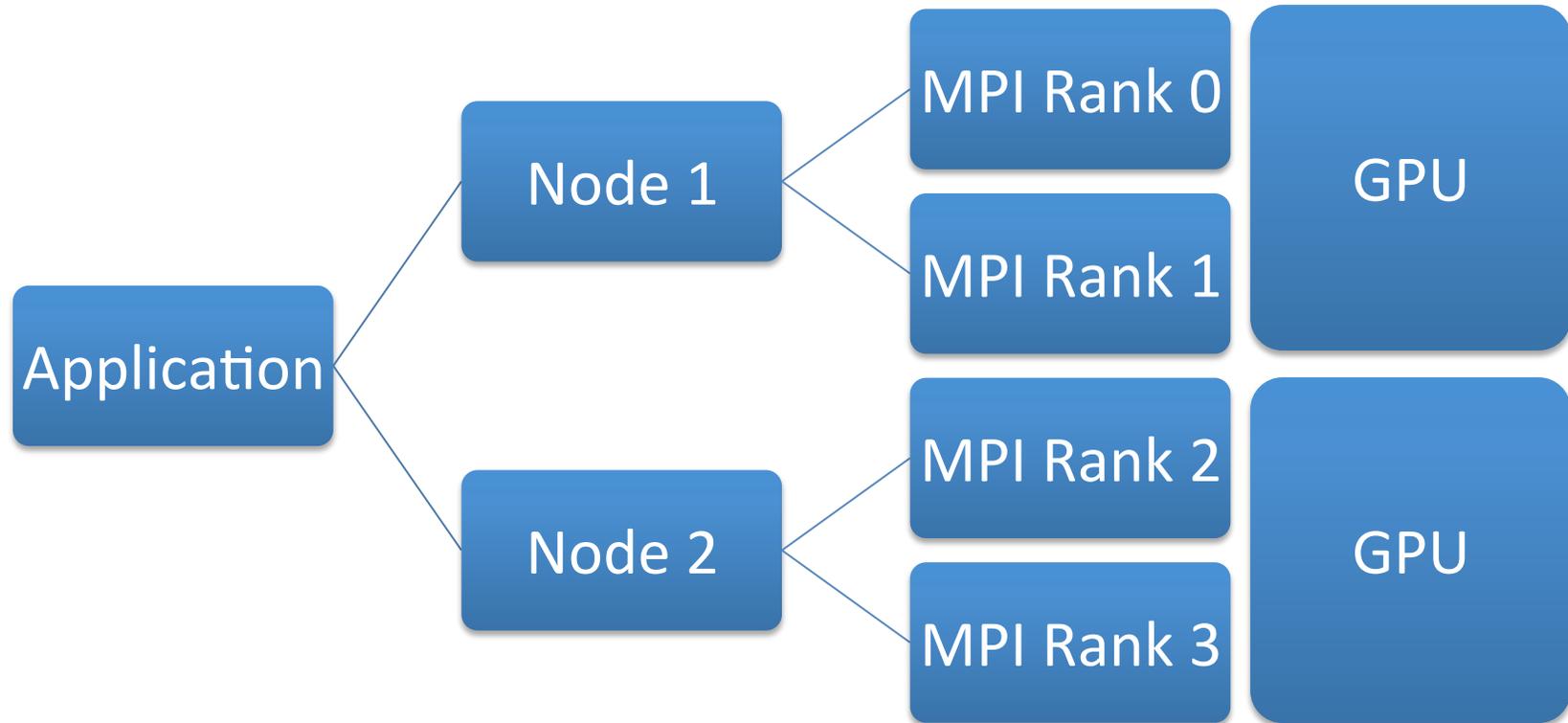
What programming models

- CPU → MPI → GPU



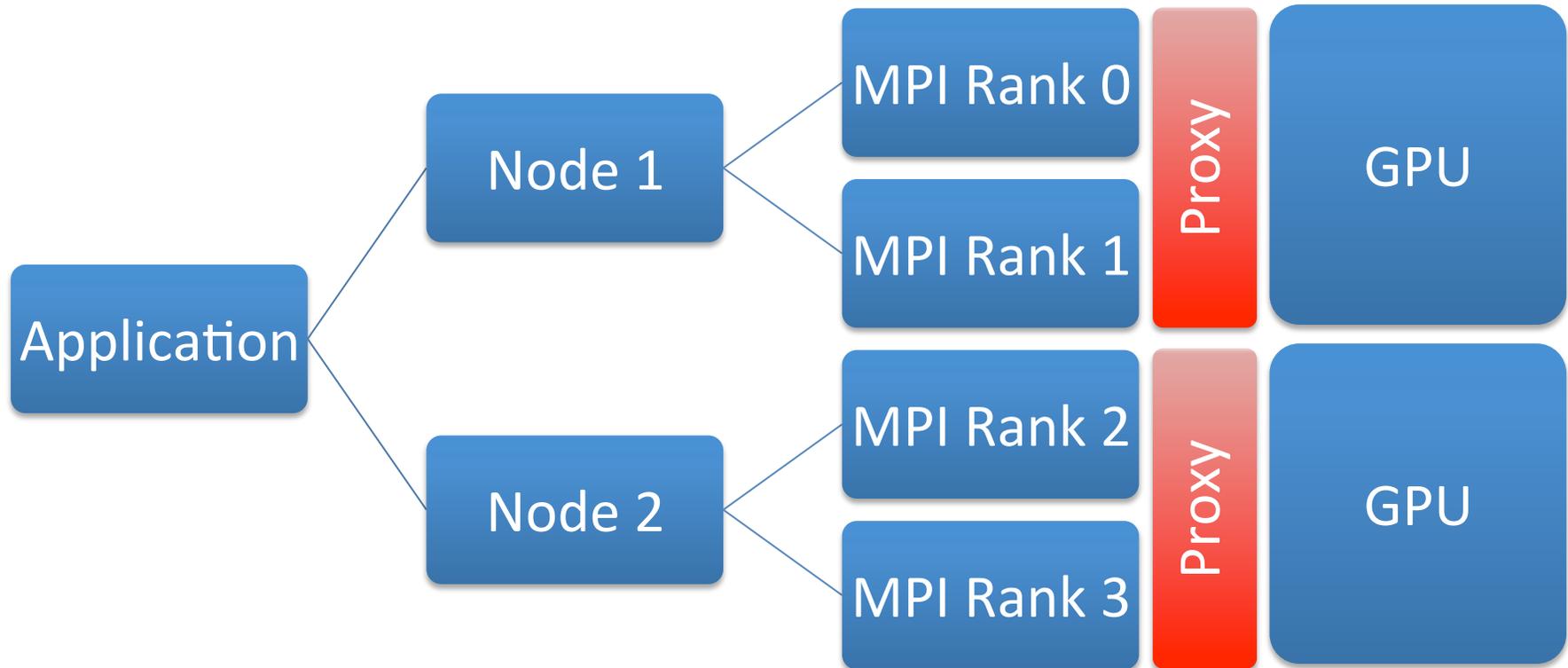
What programming models

- CPU → MPI → GPU



What programming models

- CPU → MPI → GPU



What programming models

[com/object/nvidia-kepler.nvml](http://com.object.nvidia-kepler.nvml).

Hyper-Q offers significant benefits for use in MPI-based parallel computer

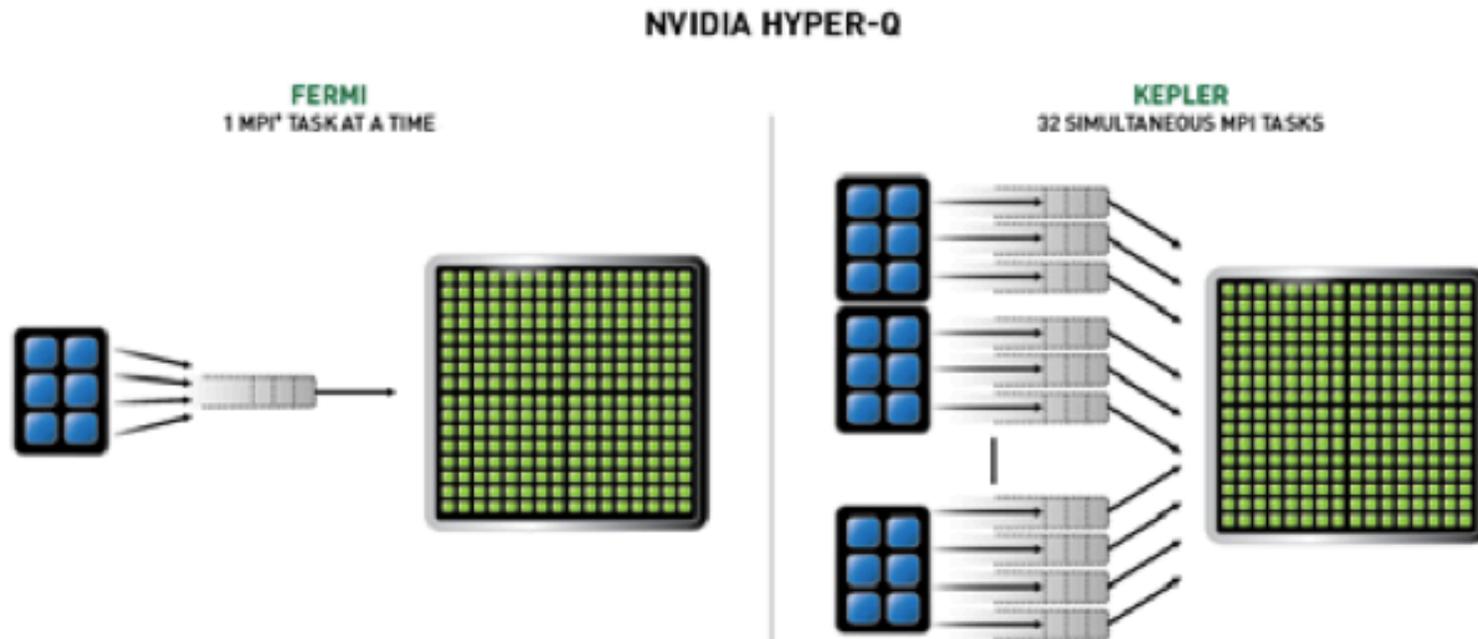


Figure 4: Hyper-Q allows all streams to run concurrently using a separate work queue. In the Fermi model, concurrency was limited due to intra-stream dependencies caused by the single hardware work queue.

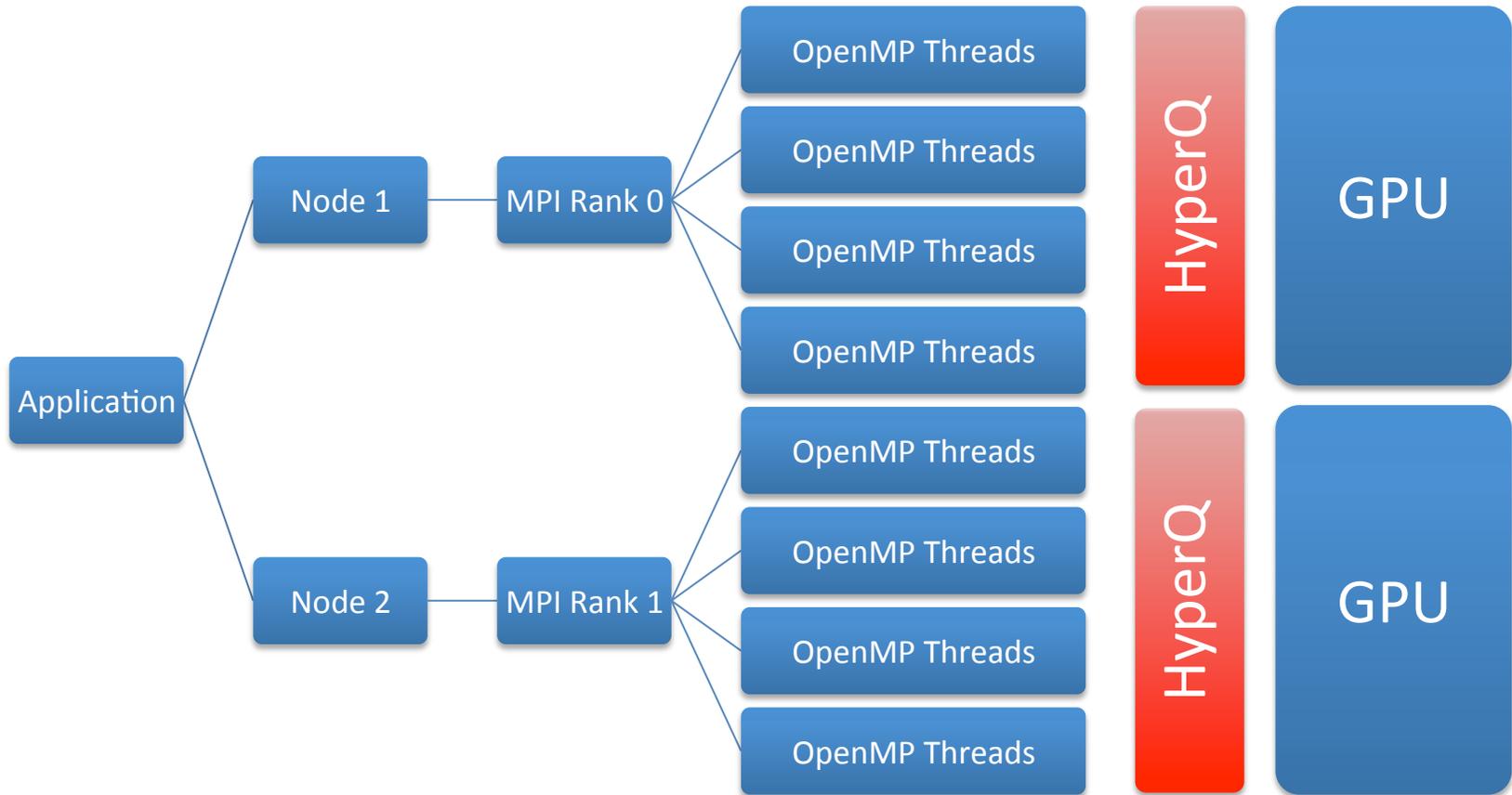
Cray Cuda Proxy

CRAY_CUDA_PROXY

Overrides the site default for execution in simultaneous contexts on GPU-equipped nodes (Hyper Q). Setting `CRAY_CUDA_PROXY` to 1 or on will explicitly enable the CUDA proxy. To explicitly disable CUDA proxy, set to 0 or off. Debugging is only supported with the CUDA proxy disabled.

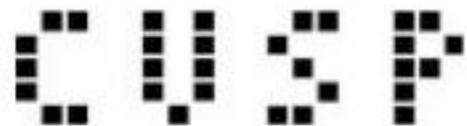
What programming models

- CPU → MPI → OpenMP → GPU



GPU Libraries

- What Libraries are available?



GPU Libraries

- What Libraries are available?

```
DefApps
adios/1.3.1(default)
adios/1.4.0
adios/1.4.1pre1
altd/1.0
atlas/3.8.4
autoconf/2.69
automake/1.12
boost/1.44.0
boost/1.49.0
bupc/2.14.2
cdo/1.5.3
cdo/1.5.5
cmake/2.8.6(default)
cula-dense/R13a
cula-dense/R14(default)
cula-dense/R15
cula-dense/R16
cula-dense/R16beta
cula-dense/R16beta5277
dataspaces/1.0.0
ddt/3.0-17491
ddt/3.1-20638
ddt/3.1-22442
ddt/3.2(default)
ddt/3.2-24924
ddt/3.2.1
dyninst/7.0.1(default)
esmf/5.2.0-p1
esmf/5.2.0r_with-lapack+netcdf_0
esmf/5.2.0rp1
esmf/5.2.0rp2(default)
esmf/5.2.0rp2_with-lapack+netcdf_0
fdupes/1.40
fpmpi/1.2
fpmpi_papi/1.2
gasnet/1.18.2
git/1.7.8
globus/5.0.4(default)
grace/5.1.22
gromacs/4.5.5
gsl/1.15
hdf5/1.8.7
hmpp/2.5.4
hmpp/3.0.0
hmpp/3.0.3
hmpp/3.0.5
hmpp/3.0.7
hmpp/3.1.0
hmpp/3.1.1
hmpp/3.2.0
hmpp/3.2.1
hmpp/3.2.3
hmpp/3.2.4(default)
hmpp-wizard-perfanalyzer/2.0
hpctoolkit/5.1.0
hypr/2.8.0b
idl/7.1
idl/8.2
lammps/14Jun12
lammps/2011-07-01
lammps/2012-01-25(default)
lammps/2012-02-17
lammps/27Jun12
liblute/1.0.0
lustredu/1.0
lustredu/1.1(default)
m4/1.4.16
magma/1.1
marmot/2.4.0(default)
matlab/7.13
matlab/7.14(default)
mercurial/1.9.2
mpip/3.3
mxml/2.6
namd/2.7b4
namd/2.8
ncl/5.2.1
ncl/6.0.0(default)
ncl/6.1.0
ncview/1.93g(default)
netcdf/3.6.2
netcdf/4.1.3
nfft/3.1.3
nvidia-sdk/4.0
nwchem/6.0
nwchem/6.1
open64/4.5.2
open64/4.5.2-klonos
open64-prgenv/4.1.20
p-netcdf/1.1.1
p-netcdf/1.2.0
p-netcdf/1.3.0
p-netcdf/1.3.1
p3dfft/2.4
parmetis/4.0.2
perfpert/2.0
pspline/1.0
python/2.7.2
r/2.14.0
ruby/1.9.3
silo/4.8
spdc/1.0.0
sprng/2.0b
starccm/5.06.010
starccm/6.06.011(default)
subversion/1.6.17
gzip/2.1
tau/2.21.4_openmp(default)
tau/2.21.4_pthread
tau/2.21_openmp
tau/2.21_pthread
tcl-tk/8.5.8
udunits/2.1.24
vampir/7.7.0-120824
vampir/8.0.0-121113(default)
vampir/8.0.0-121113-chester
vampirtrace/5.14(default)
vampirtrace/5.14-chester
vampirtrace/5.14-gpu
vampirtrace/5.14-iofs1
vasp/4.6
vasp5/5.2
vim/7.3
vmd/1.8.7
vmd/1.9
vtk/5.8
wgrib/1.8.0.13b
```

GPU Libraries

- Thrust example

```
#include <thrust/host_vector.h>
#include <thrust/device_vector.h>
#include <thrust/generate.h>
#include <thrust/reduce.h>
#include <thrust/functional.h>
#include <cstdlib>

int main(void)
{
    // generate random data on the host
    thrust::host_vector<int> h_vec(100);
    thrust::generate(h_vec.begin(), h_vec.end(), rand);

    // transfer to device and compute sum
    thrust::device_vector<int> d_vec = h_vec;
    int x = thrust::reduce(d_vec.begin(), d_vec.end(), 0, thrust::plus<int>());
    return 0;
}
```

GPU Libraries

- Thrust example

```
#include <thrust/host_vector.h>
#include <thrust/device_vector.h>
#include <thrust/copy.h>
#include <thrust/fill.h>
#include <thrust/sequence.h>
#include <iostream>

int main(void)
{
    // initialize all ten integers of a device_vector to 1
    thrust::device_vector<int> D(10, 1);

    // set the first seven elements of a vector to 9
    thrust::fill(D.begin(), D.begin() + 7, 9);

    // initialize a host_vector with the first five elements of D
    thrust::host_vector<int> H(D.begin(), D.begin() + 5);

    // set the elements of H to 0, 1, 2, 3, ...
    thrust::sequence(H.begin(), H.end());

    // copy all of H back to the beginning of D
    thrust::copy(H.begin(), H.end(), D.begin());

    // print D
    for(int i = 0; i < D.size(); i++)
        std::cout << "D[" << i << "] = " << D[i] << std::endl;

    return 0;
}
```

GPU Libraries

- Libsci_acc example

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <libsci_acc.h>
int main ( int argc, char **argv )
{
    double *A, *B;
    int *lplv;
    int n, nrhs, lda, ldb, info;
    int i, j;
    n = lda = ldb = 1024;
    nrhs = 100;
    libsci_acc_init();
    libsci_acc_HostAlloc( (void **)&A, sizeof(double)*n*n );

    lplv = (int *)malloc(sizeof(int)*n);
    B = (double *)malloc(sizeof(double)*n*nrhs);

    for ( i = 0; i < n; i++ ) {
        for ( j = 0; j < n; j++ ) {
            A[i*lda+j] = drand48();
        }
    }
    for ( i = 0; i < nrhs; i++ ) {
        for ( j = 0; j < n; j++ ) {
            B[i*ldb+j] = drand48();
        }
    }
    dgetrf( n, n, A, lda, lplv, &info);
    dgetrs('N', n, nrhs, A, lda, lplv, B, ldb, &info);

    libsci_acc_FreeHost ( A );
    free(lplv); free(B);
    libsci_acc_finalize();
}
```

GPU Libraries

- Libsci_acc example

```
dgemm('n','n',m,n,k,alpha,a,lda,b,ldb,beta,c,ldc);
```

```
dgemm_acc('n','n',m,n,k,alpha,a,lda,b,ldb,beta,c,ldc);
```

GPU Libraries

- CUSP example

```
#include<culp/hyb_matrix.h>
#include<culp/io/matrix_market.h>
#include<culp/krylov/cg.h>
int main(void)
{
    // create an empty sparse matrix structure (HYB format)
    culp::hyb_matrix<int, float,culp::device_memory>A;

    // load a matrix stored in MatrixMarket format
    culp::io::read_matrix_market_file(A,"5pt_10x10.mtx");

    // allocate storage for solution (x) and right hand side (b)
    culp::array1d<float, culp::device_memory>x(A.num_rows,0);
    culp::array1d<float, culp::device_memory>b(A.num_rows,1);

    // solve the linear system A * x = b with the Conjugate Gradient method
    culp::krylov::cg(A, x, b);

    return 0;
}
```

GPU Libraries

- CUFFT example

```
#define NX 256
#define BATCH 10
...
{
    cufftHandle plan;
    cufftComplex *data;
    ...
    cudaMalloc((void**)&data, sizeof(cufftComplex)*NX*BATCH);
    cufftPlan1d(&plan, NX, CUFFT_C2C, BATCH);
    ...
    cufftExecC2C(plan, data, data, CUFFT_FORWARD);
    cudaThreadSynchronize();
    ...
    cufftDestroy(plan);
    cudaFree(data);
}
```

GPU Libraries

- Cudatoolkit

DEVELOPER ZONE NVIDIA **CUDA TOOLKIT DOCUMENTATION**

▼ **CUDA Toolkit**

- CUDA Toolkit v5.0 Release Notes

Getting Started Guides

- CUDA Getting Started Guide for Linux
- CUDA Getting Started Guide for Mac OS X
- CUDA Getting Started Guide for Microsoft Windows

Programming Guides

- CUDA C Programming Guide
- CUDA C Best Practices Guide
- Kepler Compatibility Guide for CUDA Applications
- Tuning CUDA Applications for Kepler
- CUDA Dynamic Parallelism
- PTX ISA Version 3.1

Reference Manuals

- CUDA Runtime API
- CUDA Driver API
- CUDA Math API
- CUBLAS
- CUFFT
- CURAND
- CUSPARSE
- Thrust
- CUDA Samples

This document provides guidance on how to design and develop software that takes advantage of the new Dynamic Parallelism capabilities introduced in CUDA 5.0.

PTX ISA Version 3.1
This guide provides detailed instructions on the use of PTX, a low-level parallel thread execution virtual machine and instruction set architecture (ISA). PTX exposes the GPU as a data-parallel computing device.

Reference Manuals

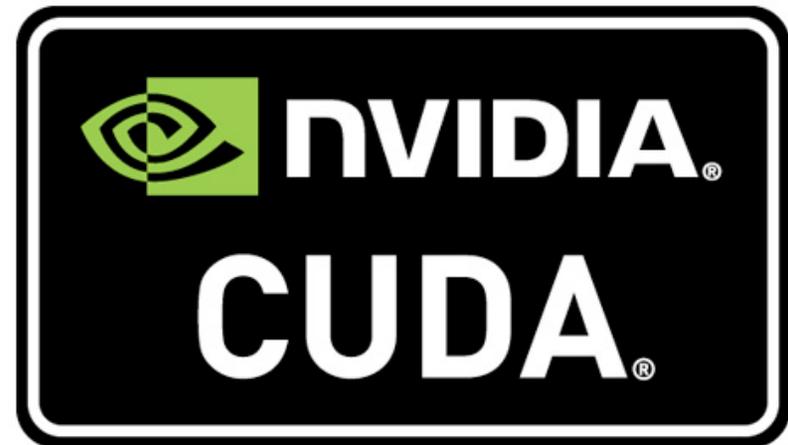
- CUDA Runtime API**
The CUDA runtime API.
- CUDA Driver API**
The CUDA driver API.
- CUDA Math API**
The CUDA math API.
- CUBLAS**
The CUBLAS library is an implementation of BLAS (Basic Linear Algebra Subprograms) on top of the NVIDIA CUDA runtime. It allows the user to access the computational resources of NVIDIA Graphical Processing Unit (GPU), but does not auto-parallelize across multiple GPUs.
- CUFFT**
The CUFFT library user guide.
- CURAND**
The CURAND library user guide.
- CUSPARSE**
The CUSPARSE library user guide.
- Thrust**
The Thrust getting started guide.

Frameworks

- OpenACC, OpenCL, CUDA, CudaFortran



OpenCL



Should you code for the hardware?

- Dynamic parallelism
- HyperQ

Conclusions